APPLICATION FOR UNITED STATES PATENT

By

Gautam Dharamshi

For

A GENERIC JAVA RULE ENGINE FRAMEWORK

A GENERIC JAVA RULE ENGINE FRAMEWORK

FIELD OF THE INVENTION

[0001] This invention relates to providing a generic Java rule engine framework that will permit the capture of events from Java objects themselves and the recording and/or processing of such events independent of application programming or object model. The present invention is particularly useful in many environments including a business-to-business electronic marketplace where business events that occur frequently may cause a need for further processing.

BACKGROUND OF THE INVENTION

[0002] Rules engines have been used to affect objects. In order to permit the application of rules to events occurring within objects, a rule engine must be capable of capturing events from the objects and programming against them. An overall architecture of a rule engine that hosts an application according to the prior art is depicted in Figure 1. Application objects 10, 11 and 12 call rule engine 20 upon the occurrence of predefined events. Rule engine 20 evaluates rule conditions, such as rule condition

50, to optimize when the rule should even be considered for execution by building RETE NET 30 using rule options expressed by the user. Rule 70 is accessed, as is rule condition 50. The rule is then executed as shown in box 60.

[0003] In order to function properly, rule engine 20 imposes certain requirements upon application objects 10, 11 and 12. First of all, rule engine 20 must be able to receive the events from application objects 10, 11 and 12 that are to be used with rule 70. Events that may be desirable to receive include field/property modification events, method calling events, object creation events and object deletion events.

[0004] In order for a programmer to be able to code rules and conditions, such as rule 70 and condition 50, that can be used against application objects 10, 11 and 12, the exact types of objects being used must be known to the programmer as well as details on the properties that are being used in the rules and conditions. The programmer must employ syntax that is able to express a wide variety of logistics while programming the rules and conditions.

[0005] Traditionally application programmers had two methods with which to accomplish this. The first is to have the rule engine define an API for the events that it

requires. Under this scenario, the application objects would fire such events explicitly when they occur.

[0006] The second method would be to have the rule engine define an object model that internally fires events automatically. Programmers of such objects would then have to ensure that each application object would adhere to the object model.

[0007] In either of these prior art solutions, programmers of objects would have to explicitly program hooks into the objects that would reach out of the object and engage the rule engine when a specific event occurred through either programming the objects to a predefined object model or to meet the API's requirements. Thus, programmers had to be aware of what the rule engines were looking for, spend time coding in the appropriate hooks and adhere to the syntax supported by the object model or API. If an unforeseen rule were to be developed for use, the programming of the objects may have to be revisited in order to update the existing hooks or add new hooks to account for the new rule. Thus, these prior art frameworks are inefficient and may cause delays in deployment of needed services due to more time-intensive programming and the possibility of needed reprogramming.

[0008] The number of Java based applications have grown considerably along with the growth of the Internet. In certain systems, events occurring within Java objects may cause a desire to take some action. One such system is a business-to-business electronic marketplace. For instance, in an electronic marketplace a shopping basket may be a Java object. When a method is executed that adds an item to that shopping basket, it may be desirable to cause some action to occur through the use of a rules engine, such as offering additional items to the shopper for possible addition to the shopping basket that relate to the newly added item.

[0009] Thus a need exists for a framework that would permit the application of Java rules through a rule engine without the need for the explicit insertions of hooks within Java objects.

SUMMARY OF THE INVENTION

[0010] An embodiment of the present invention provides a generic Java rule engine framework that permits the capture of events from Java objects themselves and the recording and/or processing of such events independent of application programming or object model.

[0011] Another embodiment of the present invention provides an electronic business-to-business marketplace having a generic Java rule engine framework that permits the capture of events from Java objects themselves and the recording and/or processing of such events independent of application programming or object model.

[0012] As such, it is an object of the present invention to permit a rule engine to be called upon the occurrence of an event within a Java object without requiring the explicit insertion of hooks into the object for invoking the rule.

[0013] It is a further object of the present invention to permit a rule engine to be called upon the occurrence of an event within a Java object on an electronic business-to-business marketplace without requiring the explicit insertion of hooks into the object for invoking the rule.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] Figure 1 illustrates the use of a rule engine upon application objects according to the prior art.

[0015] Figure 2 illustrates a generic Java rule engine framework according to an embodiment of the present invention.

[0016] Figure 3 depicts a block diagram of an electronic marketplace having a generic Java rule engine framework according to an embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0017] The present invention will be better understood by reference to the accompanying drawings.

[0018] Ideally, events could be captured from the Java objects themselves and then be processed independent of any application programming or any object model. According to an embodiment of the present invention, this can be accomplished through the use of debugging interfaces as will be described below.

[0019] JVMDI is a Sun Microsystems specification for debugging. It prescribes that any JVM that can be debugged adhere to the JPDA specification.

[0020] A VM supporting the JDPA architecture implements a JVMDI interface which defines the services that the JVM provides. It is possible to hook onto these interfaces and get events from the VM. In order to do this, it must be done in the native language using the JNI interfaces

[0021] The back-end for the JDWP (Java Debug Wire Protocol) implements this interface and the JDWP uses a

transport mechanism, either Shared Memory or Sockets, to transport these events to another JVM that is debugging the current JVM via the JDI (Java Debug Interface) specification.

[0022] The JDI defines numerous event requests. Some of those event requests support events that the rule engine would require. These events include the following: 1) Com.sun.jdi.event.ModificationWatchpointEvent, which defines an event that the JVM generates when a particular field of an object is modified; 2) Com.sun.jdi.event.BreakpointEvent, which defines a breakpoint in the JVM execution (for use in a generic framework, it can define when a particular method or all methods have been entered or exited); Com.sun.jdi.event.ClassPrepareEvent, which occurs when the class of the object is first being loaded; Com.sun.jdi.event.VMDeathEvent, which is fired when the Target JVM dies; Com.sun.jdi.event.VMDisconnectEvent, which is fired when the target JVM gets disconnected; and Com.sun.jdi.event.VMStartEvent, which is fired when the target JVM starts. Other events that the JVM fires may be useful as well.

[0023] A generic Java rule engine framework according to an embodiment of the present invention is shown in Figure

2. Hosted/Target JVM 160, the Java object from which events are to be captured, is connected to JVMDI 155. When an event occurs within Hosted/Target JVM 160, the event is fired to JVMDI 155. For instance, an item added to a shopping basket object could cause such an event.

[0024] JVMDI 155 then provides the event to JDWP 150. To tie into JVMDI 155, JDWP 150 must be implemented in the native language.

[0025] JDWP 150 conveys the event to JDI (Java Debug Interface) 145. JDWP 150 utilizes a transport mechanism, such as shared memory or sockets, to convey the event. JDI 145 provides the event to the event handler thread 120, completing the event capture. If the event is desired to be logged as a business event, logging thread 130 is also called. Logging thread 130 will then record the event in business event capture database 140.

[0026] From the event handler thread 120, the rule engine is called in main thread 125 and event definition database 135 is accessed as needed. Java provides a mechanism for dynamic class loading, so the rules could be loaded when needed.

[0027] A rule written in Java may need to adhere to an interface that the rule engine recognizes, but the internal code would be Java and the rule developer would write in

Java. The design time system could generate the actual code for the Java rule abstracting the user from the details of the context filling.

[0028] Rules are executed based on their activation by user options and when events occur on objects in an application. Conditions of the rules may need to be split into individual conditions, and their contexts bound individually to build the NET.

[0029] Upon the application of the rule in the traditional manner as discussed above in relation to Figure 1, if a change must be affected in the hosted/target JVM 160, rule callback thread 115 invokes the method to cause the change in hosted/target JVM 160 through an API 165 defined through the JDI interface.

[0030] An electronic marketplace according to an embodiment of the present invention is shown in Figure 3. Electronic marketplace 100 is shown having a generic rule engine framework that was described above with respect to Figure 2. Electronic marketplace 100 is connected to members 205, 210 and 215 and outside services 225, 230 and 235 through a computer network 200 which may be the Internet. Multiple computer networks could also be used. Hosted/target JVM Object 160 within electronic marketplace

100 could be an object used in effectuating electronic commerce.

[0031] Although the preferred embodiments of the present invention have been described and illustrated in detail, it will be evident to those skilled in the art that various modifications and changes may be made thereto without departing from the spirit and scope of the invention as set forth in the appended claims and equivalents thereof.